サイエンス社『 数値計算講義』(金子 晃)　章末課題の解答

(平成 26 年 12 月 14 日版)

第 1 章

課題 1.1　プログラム見本の `kaijo.f` あるいは `kaijo.c` およびその倍長版 `kaijo4.f` あるいは `kaijo4.c` をコンパイルし, N にいろんな値を与えて実行してみると, 様子が分かってきます. 次はこれを一気にやる参考プログラム `kadai1-1.c` の実行結果です.

```
 Result by unsigned long      Ratio     Result by unsigned long long
  1!=                   1      1.00     1!=                    1
  2!=                   2      2.00     2!=                    2
  3!=                   6      3.00     3!=                    6
  4!=                  24      4.00     4!=                   24
  5!=                 120      5.00     5!=                  120
  6!=                 720      6.00     6!=                  720
  7!=                5040      7.00     7!=                 5040
  8!=               40320      8.00     8!=                40320
  9!=              362880      9.00     9!=               362880
 10!=             3628800     10.00    10!=              3628800
 11!=            39916800     11.00    11!=             39916800
 12!=           479001600     12.00    12!=            479001600
 13!=          1932053504      4.03    13!=           6227020800
 14!=          1278945280      0.66    14!=          87178291200
 15!=          2004310016      1.57    15!=        1307674368000
 16!=          2004189184      1.00    16!=       20922789888000
 17!=          4006445056      2.00    17!=      355687428096000
 18!=          3396534272      0.85    18!=     6402373705728000
 19!=           109641728      0.03    19!=   121645100408832000
 20!=          2192834560     20.00    20!=  2432902008176640000
 21!=          3099852800      1.41    21!= 14197454024290336768
```

Ratio のところは, double に変換して一つ前の結果との比を見ています. その右は unsigned long long という 64 ビットの整数で計算したものです. これを見ると, unsigned long では $n = 13$ からおかしいですね. この他に, double の浮動小数で計算したものとの比較も可能です. どこまで正しい答えを出すか, 是非やってみてください. なお, 大体合っているかどうかのチェックなら single でも可能ですね. このことも記憶しておきましょう.

unsigned long long の方も $n = 21$ はおかしいですね. 第5 章で使い方の紹介が出てくる Risa/Asir による多倍長での計算結果は

```
rlsv:mizuka$ ~kanenko/Risa/asir
This is Risa/Asir, Version 20050209 (Kobe Distribution).
......
[221] fac(20);
2432902008176640000
[222] fac(21);
51090942171709440000
```

となりました. なお, 著者の FreeBSD on Athlon x64 機で実行したところ, unsigned long は unsigned long long と同じ結果となりました. 64 ビットの CPU ならその方が自然ですね. このように long

の長さは機械によって異なるので注意しましょう．自分の計算機の CPU が 32 ビットか 64 ビットか，これを使って是非実験してみてください．

講義時のレポートとしては，符号付き long (FORTRAN 77 の INTEGER*4 も同じ) でもやってもらいましたが，このときはどの程度増加しているかを見ずに，オーバーフローの結果値が負になったところでおかしいと答えた人がかなり居ました．また符号無し long の場合は，更に計算を続けると，$n = 34$ で 0 になるので，$n = 33$ まで正しいと答えていた人がかなりいました．これらの判断はいずれも甘すぎます．なお，どうして unsigned long で $33! = 2147483648$ (計算機が答えた間違った値) に更に 34 を掛けたら，結果がきれいに 0 になったのか，興味の有る人は是非理由を考えてみましょう．

倍精度実数によるプログラム見本の `kaijodbl.f`, `kaijodbl.c` の計算結果は一部のみ示します．

```
gcc kaijodbl.c -o kaijodbl
./kaijodbl
Give n:21
21!=51090942171709440000.000000
./kaijodbl
Give n:22
22!=1124000727777607680000.000000
./kaijodbl
Give n:23
23!=25852016738884978212864.000000
```

最後の結果は明らかにおかしいですね (10 で割れなくなっています)．22! は一見もっともらしいのですが，数字が十進で 18 個も並んでいるのは，倍精度実数の有効桁数を考えるとちょっと心配です．試しに Risa/Asir でやってみると

```
[0] fac(22);
1124000727777607680000
[1] fac(23);
258520167388849766400000
```

で 22! はかろうじて合っています．この後を続けても答の桁数は合っていますが，有効桁数を越えた部分の数字は信用してはいけません．値が出たからといって，そのまま信じるととんでもないことになります．

課題 1.2 (略)

課題 1.3 いろいろな n についてそこまでの和を求めて十進法で出力し，その数値が目で見て一致したところとして $n = 2097151$ から変わらなくなったと答えた人が多かったのですが，これは少し危険です．計算機の内部は二進法なので，もしかしたら微妙に異なっている二つの数が十進の出力フォーマットでたまたま一致しただけかもしれません．やはり，一つ前の計算結果との差をとり，それが 0 と判断されたところを答えるようにしましょう．今回はこの方法でも，確かに $n = 2097151$ から一定となりました．詳しくいうと，$i = 1$ から n までの $1/i$ の和の値を $f(n)$ とするとき，$f(2097150) < f(2097151) = f(2097152) = \cdots$ ということです．解答例 kadai1-3.c は，このときの n の前後の値での内部表現を覗いて確認しています．また，同じ範囲の $1/i$ の和を逆に加えたもの

も示しています，倍精度で同じ範囲を加えると，逆に加えたものの方が確かに正確になっていることが分かりますね．次は kadai1-3.c の実行結果です．

```
./kadai1-3
Stabilized to the value 15.403683 at N = 2097151
2097150:15.403682  7B 75 76 41
2097151:15.403683  7C 75 76 41
2097152:15.403683  7C 75 76 41
Result of backward summation = 15.132898
Sum of the same range via double precision =     15.133306218241035
```

最初は，ほとんどの人が，頭から加えたものの方が情報落ちのために小さな値になると思うのですが，この結果は逆ですね． その理由は，和が一定になる直前に何が足されているかを考えれば分かります．$\frac{1}{n}$ は大きな n では n が $n+1$ になってもほとんど変わりません．最後は二進法の四捨五入で，最後の桁の一つ下が 1 のものが繰り上がって，最後の桁に 1 が加え続けられているに違いないのです．つまり1 回ごとに，ほぼ倍に近いものが足されていることになります．逆から加えるとこれは当然，ほぼ正確に足されますね．

級数の足し算のループの変数を float i にしている人がいましたが，これは勧められません．今回の問題では特に不都合は生じないのですが，一般にはきちんと指定回数だけ回すには整数変数でループを制御するようにと言われています．

課題 1.4　(略)

課題 1.5　(略)

課題 1.6　(略)

第 2 章

課題 2.1　サポートページに置かれたプログラム zeta2.f または zeta2.c による $s_N = \sum_{i=1}^{N} \frac{1}{i^2}$ の計算結果を見ると，

N	s_N
10	1.549767731166541
100	1.634983900184892
1000	1.643934566681561
10000	1.644834071848065
100000	1.644924066898242
1000000	1.644933066848770
10000000	1.644933966847260

で，真の値の近似値 $\frac{\pi^2}{6} = 1.64493406684822643647\cdots$ と比較すると，確かにちょうど $\frac{1}{N}$ のところに誤差が出ていますが，そこに $\frac{1}{N}$ を加えてやると，そのほぼ倍の桁まで正しい数字が並んでいることが観察されます．これは，

$$\frac{\pi^2}{6} = s_N + \frac{1}{N} + O(\frac{1}{N^2})$$

という式が成り立っているからだと予想されます．実際，教科書 p.47, 例 2.1 に与えた打ち切り誤差の評価式から

$$\frac{1}{N+1} \leq \frac{\pi^2}{6} - s_N = \sum_{i=N+1}^{\infty} \frac{1}{i^2} \leq \frac{1}{N}$$

よって，このすべての辺から $\dfrac{1}{N}$ を引けば，

$$-\frac{1}{N(N+1)} \leq \frac{\pi^2}{6} - \left(s_N + \frac{1}{N}\right) \leq 0 \tag{1}$$

で，上の予想が確かめられました．なお，参考までに，教科書の p.54, 例 2.4 の式変形を用いると，更に精密に

$$\begin{aligned}
\frac{\pi^2}{6} - s_N &= 1 - \sum_{i=1}^{N} \frac{1}{i(i+1)} + \sum_{i=N+1}^{\infty} \left(\frac{1}{i^2} - \frac{1}{i(i+1)}\right) \\
&= \frac{1}{N+1} + \sum_{i=N+1}^{\infty} \frac{1}{i^2(i+1)} \\
&= \frac{1}{N+1} + \sum_{i=N+1}^{\infty} \left(\frac{1}{(i-1)i(i+1)}\right) - \sum_{i=N+1}^{\infty} \left(\frac{1}{(i-1)i(i+1)} - \frac{1}{i^2(i+1)}\right) \\
&= \frac{1}{N+1} + \frac{1}{2}\left(\frac{1}{N} - \frac{1}{N+1}\right) - \sum_{i=N+1}^{\infty} \frac{1}{i^2(i^2-1)} = \frac{1}{N} - \frac{1}{2N^2} + O\left(\frac{1}{N^3}\right)
\end{aligned}$$

という漸近展開が得られます．つまり，計算した近似値に $\dfrac{1}{N}$ を加え，更に $\dfrac{1}{2N^2}$ を引けば，有効桁数は3 倍ほどに延びるという訳です．上の表でこのことを観察してみましょう．

実際の演習では，残念ながら理由を正しく書けた人はそう多くありませんでした．理由を “丸め誤差のため” などと意味不明なことを書いていた人が結構いましたが，このように，有効桁の途中に最初から現れるのは打ち切り誤差以外には考えられません．丸め誤差というのは，端のほうからじわじわと大きくなってくるものです．これは，この『数値計算講義』でもこれから繰り返し現れるテーマの一つです．

課題 2.2 (略)

課題 2.3 まずはプログラミングですが，見本の `sintaylor.f` か `sintaylor.c` でやっている $\sin x$ の Taylor 展開の真似をすれば簡単にできます．ただ，一般項を $t = \dfrac{(-1)^{n-1}x^n}{n}$ としてしまうと，

```
s=s+t;
t=-t*x*i/(i+1);
```

などと無駄な掛け算が必要になります．ここでは，プログラミング上の一般項は $t = (-1)^{n-1}x^n$ にとり，

```
s=s+t/i;
t=-t*x;
```

とするのがエレガントです．C と FORTRAN の見本を `kadai2-3.c`, `kadai2-3.f` として置きました．

N	Partial sum	Value modified by 1/2N
10	0.645634920634921	0.695634920634921
100	0.688172179310195	0.693172179310195
1000	0.692647430559822	0.693147430559822
10000	0.693097183059958	0.693147183059958
100000	0.693142180584982	0.693147180584982
1000000	0.693146680560253	0.693147180560252
10000000	0.693147130560106	0.693147180560106
100000000	0.693147175560419	0.693147180560419
1000000000	0.693147180060647	0.693147180560647
True value:	0.693147180559945	

誤差の考察ですが，$\log(1+x)$ の Taylor 展開を最も易しく導くには，等比級数

$$\frac{1}{1+x} = 1 - x + x^2 - + \cdots + (-1)^{N-1}x^{N-1} + (-1)^N \frac{x^N}{1+x}$$

を一回不定積分すればよい：

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - + \cdots + (-1)^{N-1}\frac{x^N}{N} + (-1)^N \int_0^x \frac{t^N}{1+t}dt$$

昨年はこれを実習の時間にホワイトボードで説明しましたが，今年は連休をはさんで2 週間有ったので，自分で挑戦してもらいました．この級数は $x=1$ のときも最後の項の絶対値が $\leq \frac{1}{N}$ なので，級数はぎりぎりで収束し，$\log 2$ を与えます．(一般論としては Abel の定理という，微積でもかなり高級なものですが，この例だけなら高校生でも理解できるでしょう.) $x=1$ に限定して剰余項をもう少し正確に見積もると

$$(-1)^N \int_0^1 \frac{t^N}{1+t}dt = (-1)^N \Big[\frac{t^{N+1}}{N(1+t)}\Big]_0^1 - (-1)^N \int_0^1 \frac{t^{N+1}}{N(1+t)}dt = (-1)^N \frac{1}{2N} + O\big(\frac{1}{N^2}\big)$$

となるので，これを部分和に加えると，部分和の最後の項が半分になり，それで誤差が $O\big(\frac{1}{N^2}\big)$ になるという訳です．

実際の演習における事例として，$\log 2$ の計算で，$N = 10^n$ まで足したとき，コンピュータの出力ではちゃんと小数点以下 $n+1$ 位に 5 を足すと真値により近づくことを示しているのに，“前問と同様打ち切り誤差の主要項は 1/N” などと書いたやってることと言ってることがちぐはぐな人がいましたが，この場合は, 上の計算からも分かるように，級数を $\frac{(-1)^{N-1}}{N}$ で打ち切ったときの打ち切り誤差の主要項は $\frac{(-1)^N}{2N}$ ということになります．ここでは $N = 10^n$ は偶数なので，実際にこの値を加えることになります．

最後の項の分母の N を $2N$ に変えるというのを勘違いして，N 項目まで足すのを，$2N$ 項目まで足すというのと混同していた人がいましたが，単に $2N$ 項目まで足した場合は，打ち切り誤差は $O(1/2N) = O(1/N)$ で大して変わりません．ほぼ半分になるだけです．0.5 の半分は 0.25 なので，誤差が半分になっただけでは有効数字は1 桁も増えません．

なお，$1/2N$ を調節すると，有効桁数が倍に延びるという現象は、N が大きくなると，次第になまってきます．これが “丸め誤差の影響” なのです．

第 3 章

課題 3.1 (略, 解答例はプログラム見本の `stirling.f`, `stirling.c` にあります．いずれも参考までに漸化式を少し伸ばしたものの出力値を提供しています．ただし，最初に置いていた `stirling.f` は 2 項用いたものの精度が理論値まで出ていませんでした．`stirling.c` はこの原因究明のために作成したものですが，原因は `PI=3.14159265358979323846` などと書いていたためで，これは `PI=3.141592653589793238460D0` などと書かないと，ここで単精度に落ちてしまいます．しばらく Fortran を使っていなかったので気づくのに時間がかかってしまいました (^^;.

課題 3.2 実際に演習で課題として課した $\mathrm{Arcsin}\, x$ について，解答例と講評を示します．教科書にも掲載されているプログラム例 `suchibibun.f` あるいは `suchibibun.c` の中の3 個所の `sin` を `asin(x)` に取り替え, 真値の `cos(a)` を `1/sqrt(1-a*a)` に変えるだけで難しいところはありません．(真値を変更するのを忘れてそのまま提出した人が居ましたが，何のために真値を表示させているのか考えましょう！) なお，メッセージも忘れずに修正しましょう．数値計算結果には影響はありませんが，プログラマとしては忘れてはいけません．解答見本は `kadai3-2-1.f` と `kadai3-2-1.c` です．

問題文からは，導函数を計算する点はどこでも良いのですが，$a = 0.5$ などが多かったです．変更せずに $a = 1$ を使っちゃうと nan が出ますが，これは非数値 non-arithmetic number の略です．実際，$\mathrm{Arcsin}\, 1$ はこの函数のグラフ (書けない人が居たのがちょっと心配ですが (^^;) から，数学的には ∞ となることが分かるので，適切な選択ではありません．

実験に対する論評はプレゼンの $\sin x$ に対するものの丸写しで OK です. nan が出力されているのにそのまま提出しちゃった人は，論評の仕様が無かったですね．猛反省しましょう．なお，FORTRAN の出力に `**********` と出たのは，`F22.16` のフォーマットで書ききれない数値になったときには長さだけの星を出力するという FORTRAN の慣習です．

フォーマットで苦労した人も居たようですが，例えば

```
C 言語:      printf("%2s%22.15lf%4s%22.15lf%c","x=",x,", y=",y,'\n')
に対応して
F77:          WRITE(*,100)'x=',x,', y=',y
          100 FORMAT(1H ,A2,F22.15,A4,F22.15)
```

となります．(言語仕様の差により，改行コードの出力は完全には同値ではありません．) C 言語のときは，普通は上のようには書かず,

```
printf("x=%22.15lf, y=%22.15lf\n",x,y)
```

のように書いてしまいますが，同じことです．

2 階微分の方は `suchibibun2.f` あるいは `suchibibun2.c` を書き直せばよろしい．これも `DSIN` を `DASIN` に書き直し，真値とメッセージを適当に修正するだけです．解答は `kadai3-2-2.f` および `kadai3-2-2.c` として置きました．

微積を習ってから 2 年経って，真値である $\mathrm{Arcsin}\, x$ の 2 階微分が計算できなくなった人は，院試に備えて復習した方がいいですよ (*^^*).) どうしても自分でできなかったら，演習としては Risa/Asir を立ち上げて,

```
[0] X=asin(x);
asin(x)
```

```
[1] diff(X,x);
((-x^2+1)^(-1/2))
[2] diff(@@,x);
((-x^2+1)^(-3/2))*x
[3] quit;
```

とやるとよい．つまり答は $\dfrac{x}{\sqrt{1-x^2}^3}$) となることが分かります．

あるいは maxima を立ち上げて

```
(%i1) diff(asin(x),x,2);
                                        x
(%o1)                             -----------
                                        2 3/2
                                  (1 - x )
(%i2) quit();
```

としてもよい．

2 階微分の実験結果の考察も，$\sin x$ に対する教科書の解説と同様ですが，精度が 8 桁程度しか出ていないことをよく認識してください．

課題 3.3 (略，解答見本は教科書に書いたように `2kai4ji.f` の名前でプログラム見本のところに置いてあります．)

課題 3.4 (略，同上 `hasibibun.f`)

第 4 章

課題 4.1 問題文の中に挙げられた $\dfrac{1}{x^2+1}$ (プログラム例 `kadai4-1a.f`) と $\dfrac{1}{x^3+1}$ (プログラム例 `kadai4-1b.f`) でやってみると，同じように見えて結果がひどく異なることが分かります．すなわち，後者では

$h=\dfrac{1}{N}$	台形公式	Simpson 公式	
0.100000000000000	0.835022755845639	0.835653194557389	
0.010000000000000	0.835642598155347	0.835648848702195	
0.001000000000000	0.835648785764711	0.835648848264765	
0.000100000000000	0.835648847639747	0.835648848264745	← Simpson の最良
0.000010000000000	0.835648848258767	0.835648848265024	
0.000001000000000	0.835648848264674	0.835648848264700	← 台形の最良
0.000000100000000	0.835648848311256	0.835648848311306	

真の値：$\dfrac{1}{3}\log 2+\dfrac{\pi}{3\sqrt{3}}=0.835648848264721$

と，ほぼ理論通りに台形公式の誤差が $O(h^2)$, SImpson 公式の誤差が $O(h^4)$ で増えて行くのに対し，

前者の方は，

$h=\frac{1}{N}$	台形公式	Simpson 公式	
0.100000000000000	0.784981497226790	0.785398153484804	← Simpson が7 桁正しい
0.010000000000000	0.785393996730782	0.785398163397439	← Simpson が 13 桁正しい
0.001000000000000	0.785398121730781	0.785398163397447	
0.000100000000000	0.785398162980807	0.785398163397474	
0.000010000000000	0.785398163393475	0.785398163397639	
0.000001000000000	0.785398163397771	0.785398163397802	

真の値: $\frac{\pi}{4}=0.785398163397448$

と最初から Simpson 公式の精度が異常に良く, まるで 6 次の近似式のように見えます．この異常性を発見してもらうのがこの問題の一つの主題でしたが，気づいた人はあまり多くはありませんでした．というか，多くの人は，まだ合っている桁が有るにも拘わらず，理論的に予想される桁で線を引き，“ほぼ理論の予想通り” と答えていました．そういう感動の無い生活はさびしいですね．(^^;

この異常現象の理由は以下のとおりです: 一般に Simpson 公式は (もし被積分函数がいくらでも微分できれば),

$$\mathrm{Simp}[f]-\int_a^b f(x)dx=e_4[f,a,b]h^4+e_6[f,a,b]h^6+\cdots$$

という誤差の漸近展開を持ちます．ここで，係数は f と a,b に依存して決まる汎函数ですが，$f=\frac{1}{x^2+1}$ で $[a,b]=[0,1]$ のとき，たまたま $e_4[f,a,b]=0$ となるのです．この誤差の漸近展開と係数の決定は，高橋-森の理論と言う，函数論を使って導く誤差評価式の例であり，かなり高級な内容なので，ここではその導出はやりませんが，注意すべきことは，この例に対しても，一微小区間分では Simpson 公式は理論通りちょうど h^5 の誤差を持っており，総和して初めてそれが消えるということです．試しに，同じ被積分函数で $[1,2]$ 上の定積分を計算してみてください．どうなるでしょうか?

実は，定積分 $\int_0^1\frac{1}{x^2+1}dx$ に対する Simpson 公式の計算例は，昔高校の教科書にも載っていたものですが, Simpson 公式の誤差を見るには，上の理由によりはなはだ不適切なものです．

今回も真の値を取り替えるのを忘れて，違う問題の値をそのまま使い，あまつさえ “理論通りの速度で真の値に近づいていった” なんて書いていた人が居ました．善意に解釈すれば，実用的な問題では真の値は不明なことが多いので，近似値の各桁の数字の動き方から判断したのだと思いますが，それならそうと書きましょう．$\int_0^1\frac{dx}{x^3+1}$ の真値の求め方については，質問が出てホワイトボードで計算してあげたこともありますが，出なかった年も，みんな分かってるという訳でもなかったようです．念のため $\int_0^1\frac{1}{x^3+1}dx$ の真値の計算を書いておきます．

$$\begin{aligned}\int_0^1\frac{1}{x^3+1}dx&=\int_0^1\frac{1}{3}\Big(\frac{1}{x+1}-\frac{x-2}{x^2-x+1}\Big)dx\\&=\Big[\frac{1}{3}\log(x+1)-\frac{1}{6}\log(x^2-x+1)\Big]_0^1+\frac{1}{2}\int_0^1\frac{dx}{x^2-x+1}\\&=\frac{1}{3}\log 2+\frac{1}{2}\int_0^1\frac{dx}{(x-\frac{1}{2})^2+\frac{3}{4}}=\frac{1}{3}\log 2+\frac{1}{2}\Big[\frac{2}{\sqrt{3}}\operatorname{Arctan}\frac{2}{\sqrt{3}}\big(x-\frac{1}{2}\big)\Big]_0^1\\&=\frac{1}{3}\log 2+\frac{2}{\sqrt{3}}\operatorname{Arctan}\frac{1}{\sqrt{3}}=\frac{1}{3}\log 2+\frac{2}{\sqrt{3}}\frac{\pi}{6}=\frac{1}{3}\log 2+\frac{\pi}{3\sqrt{3}}\end{aligned}$$

maxima でも次のように求まります:

```
(%i1) integrate(1/(x^3+1),x,0,1);
                          6 log(2) + sqrt(3) %pi    %pi
(%o1)                     ---------------------- + ------
                                   18                3/2
                                                   2 3
(%i2) float(%o1);
(%o2)                      0.835648848264721
```

maxima の出力結果は同類項がまとめられていないですが，まあこんなもんでしょう．もっと多くの桁が見たいときは，`fpprec: 100; bfloat(%o1)` とかを実行します．

課題 4.2　図を見れば分かるように，この和は，ちょうど真中付近に高さをとった和に相当するので，突き出した三角形と引っ込んだ三角形が打ち消し合って台形公式とほぼ同様の精度を示すことが予想される．講義の例と同じ $\displaystyle\int_0^1 \frac{1}{x+1}dx$ で実験してみると確かにそうなる：

$h=\dfrac{1}{N}$	$\displaystyle\sum_{i=1}^{N} f((i-0.5)h)h$	
0.100000000000000	0.692835360409960	
0.010000000000000	0.693144055628301	
0.001000000000000	0.693147149309952	
0.000100000000000	0.693147180247461	
0.000010000000000	0.693147180556893	
0.000001000000000	0.693147180560284	⟵ ここから崩れる
0.000000100000000	0.693147180575947	
0.000000010000000	0.693147180433543	

True value : 0.693147180559945

FORTRAN プログラムは kadai4-2.f 参照. この理由を数式で示すには，区間一つ分での誤差を Taylor 展開を使って計算してみれば良い．

$$E = f(\frac{h}{2})h - \int_0^h f(x)dx = \{f(0) + \frac{f'(0)}{2}h + O(h^2)\}h - \int_0^h \{f(0) + f'(0)x + O(x^2)\}dx$$
$$= \{f(0)h + \frac{f'(0)}{2}h^2 + O(h^3)\} - \{f(0)h + f'(0)\frac{h^2}{2} + O(h^3)\} = O(h^3)$$

よって全誤差は $O(h^3) \times \dfrac{1}{h} = O(h^2)$ で，確かに2 次の近似式となっている．つまり，Riemann 近似和でも代表点の取り方によっては台形公式と同じ 2 次の精度が出せる．

上の評価を Taylor 展開の中心を $\dfrac{h}{2}$ にし，剰余項付きでもう少していねいにやると，

$$E = f(\frac{h}{2})h - \int_0^h f(x)dx = f(\frac{h}{2})h - \int_0^h \{f(\frac{h}{2}) + f'(\frac{h}{2})(x-\frac{h}{2}) + \frac{f''(\theta)}{2}(x-\frac{h}{2})^2\}dx$$
$$= -\int_0^h \frac{f''(\theta)}{2}(x-\frac{h}{2})^2\}dx$$

よって $|f''(x)| \le M_2$ と仮定すれば，

$$|E| \le \frac{M_2}{2}\int_0^h (x-\frac{h}{2})^2 dx = \frac{M_2}{2}\cdot\frac{2}{3}\Big(\frac{h}{2}\Big)^3 = \frac{M_2}{24}h^3$$

この評価は台形公式の誤差評価 $\dfrac{M_2}{12}h^3$ の半分なので，もっと愛用されてよさそうなのに，世の中では専ら台形公式が使われている理由の最大のものは，恐らく，実際に数値積分を使う現場では，被

積分関数の格子点での値だけが与えられており，中間点での関数の値は補間計算によりわざわざ求めなければならず，余計な手間がかかるためでしょう．

課題 4.3 （ 略，真値は $\frac{\pi}{2e}$, 実装例は `kadai4-3.f`.）

課題 4.4 `kadai4-4.f` を用いて以下のような実験をしてみました．

(1) 区間 $[0, 10000]$ に限って台形公式で直接計算した値，

N	$h = \frac{1}{N}$	$\int_0^{10000} \frac{\sin x}{x^2+1} dx$
1000000	0.100000000000000	0.645926822025221
10000000	0.010000000000000	0.646752798870661
100000000	0.001000000000000	0.646761048968107
1000000000	0.000100000000000	0.646761131467990

(2) 変数変換 $x = t^2$ により，$\int_0^{\infty} \frac{2t \sin t^2}{t^4+1} dt$ に直して，区間 $[0, 10000]$ 上で台形公式により計算してみた値，

N	$h = \frac{1}{N}$	$\int_0^{10000} \frac{2x \sin x^2}{x^4+1} dx$
1000000	0.100000000000000	0.646784257509133
10000000	0.010000000000000	0.646761234093508
100000000	0.001000000000000	0.646761122661109
1000000000	0.000100000000000	0.646761122779331

(3) 部分積分を2 回行い，

$$\begin{aligned}\int_0^{\infty} \frac{\sin x}{x^2+1} dx &= \Big[-\frac{\cos x}{x^2+1} \Big]_0^{\infty} - \int_0^{\infty} \frac{2x \cos x}{(x^2+1)^2} dx \\ &= 1 - \Big[\frac{2x \sin x}{(x^2+1)^2} \Big]_0^{\infty} + 2 \int_0^{\infty} \Big(\frac{-4x^2}{(x^2+1)^3} + \frac{1}{(x^2+1)^2} \Big) \sin x dx \\ &= 1 + 2 \int_0^{\infty} \frac{1-3x^2}{(x^2+1)^3} \sin x dx\end{aligned}$$

と変形し , 最後の積分を区間 $[0, 1000]$ 上で台形公式により計算してみた値，

N	$h = \frac{1}{N}$	$1+2\int_0^{1000} \frac{1-3x^2}{(x^2+1)^3} \sin x dx$
10000	0.100000000000000	0.645084048663225
100000	0.010000000000000	0.646744455087953
1000000	0.001000000000000	0.646760956115702
10000000	0.000100000000000	0.646761121116145
100000000	0.000010000000000	0.646761122771527

(4) 変数変換 $x = e^t$ により，$\int_{-\infty}^{\infty} \frac{e^t \sin e^t}{e^{2t}+1} dt$ に直して，区間 $[-20, 20]$ 上で台形公式により計算してみた値，

N	$h = \frac{1}{N}$	$\int_{-20}^{20} \frac{e^x \sin e^x}{e^{2x}+1} dx$
400	0.100000000000000	0.642000741386176
4000	0.010000000000000	0.646578200932416
40000	0.001000000000000	0.646760774679769
400000	0.000100000000000	0.646761065323832
4000000	0.000010000000000	0.646761120020722
40000000	0.000001000000000	0.646761122803079

この問題には真の値というものはありませんが，これらの計算結果に打ち切り誤差と丸め誤差を加味して，0.64676112 までは正しいと予想されます．なお，現在最も進んでいると思われる高橋-森-杉原-大浦の2 重指数積分公式による計算結果は，0.646761122779130 を与えます (`kadai4-4omake.f`).)

この問題は例年二三人が解くだけですが，2〜3 桁くらいなら誰でも何とか求まるでしょう！適当なところで切ってそのまま Simpson 公式を適用しても 4 桁くらいはすぐ求まるようです．もうちょっと気合いを入れると 8 桁くらいまで行くようです．最初の実習の年は 0.64676112 まで正しく答えた人が二人居ました．あまり親切にプログラム見本を配らない方が教育上良いかもしれませんね．もっとも教員の方も演習終了時間を 1 時間過ぎても帰れなくて大変でしたが (^^;.

どのくらいの区間に切り詰めるかは捨てた部分の誤差の大きさを見積もって決めるのです．区間を大きくすれば無限区間を短縮したための打ち切り誤差は小さくなりますが，分割数を大きくしないと，数値積分公式の打ち切り誤差が大きくなります．区間 [0.100000] で分割数 1000 などという計算をしないように！ 分割数があまり大きいと丸め誤差の累積が無視できなくなります．という訳で，全く工夫せずにもとのままの函数を積分する場合でも，両者の兼ね合いを見て，区間と分割数を最適に決めるのは，とても良い練習になると思います．

課題 4.5 （ 略，実装例はプログラム見本の第 6 章 `sekibun.f`,`sekibun.c`に含まれています．EXTERNAL とか函数ポインタが使われていますが，サブルーチンだけを抜き出し，それに直接被積分函数を書き込んで初等的に使うこともできます.）

課題 4.6 この問題は，数値計算の教科書などで，よく,『 0.1 は二進小数では無限小数になるので，計算機に入れると有限で切られてしまい，10 個加えても 1 よりほんの少し小さくなってしまう．よってこのようなプログラムを書いてしまうと，10 回反復するつもりが 11 回実行されていまうので, 十分注意するように』と書かれているものをかつて単に『 このプログラムを走らせたらどうなるか』という風に期末試験に出したことが有るのですが，監督しながら考えていて，待てよ本当にそれで良いのだろうか？と考え込んでしまったものです．

実際に教科書の課題のところに書かれたプログラムを打ち込んで（ 面倒な人はプログラム見本の第 4 章のところに置いてある同名のものを利用してください）実行してみてください．ちゃんと 1.10 が出力されるでしょ？ 言われている通りですね．でもちょっと待ってください．0.1 は四捨五入の二進数版により，`3d cc cc cd` と最後が繰り上がって，真の 0.1 よりは逆に少し大きくなっています．実際にも，次のような，途中結果のメモリデータを出力するプログラム `kadai4-6.c` を走らせてみると，確かにそうなっています．だったらこれを 10 個加えたら，1 より少し大きくなってループは停止し，1.00 が返りそうですが，ではなぜ 1.10 になったのでしょう．下のプログラムを実行してみると分かるように，このループが計算しているのは，最初は $0.0 + 0.1$, 2 回目は $0.1 + 0.1$, とこのあたりまでは確かに，正しい値より大きめになっているのですが，$0.2 + 0.1$, $0.3 + 0.1$ と進むにつれて，情報落ちのための四捨五入の二進数版が複雑に生じ，ついには真の値よりも小さくなっています．この問題はここまで見ないと正しく答えられないのでした．試験中に『 この問題は思ったより奥が深いぞ』とつぶやいたのは，このためです．コンピュータの話はちゃんと実験して確かめてからでないとうっかりしたことは言えませんね (^^;.

kadai4-6.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void dumps(double *y){
  int i;
  unsigned char *p;
  p=(unsigned char *)y;
  for (i=0;i<8;i++){
    printf("%02x ",*(p+i));
  }
  printf("\n");
}
int main (void){
  double s;
  int i=1;
  s=(double)0;
  do {
    s=s+0.1;
    printf("i=%d,s=%5.2lf,memory dump:",i,s);
    dumps(&s);
    i++;
  } while (s<(double)1);
  printf("%5.2f\n",s);
  return 0;
}
```

実行結果

```
i=1,s= 0.10,memory dump:3f b9 99 99 99 99 99 9a
i=2,s= 0.20,memory dump:3f c9 99 99 99 99 99 9a
i=3,s= 0.30,memory dump:3f d3 33 33 33 33 33 34
i=4,s= 0.40,memory dump:3f d9 99 99 99 99 99 9a
i=5,s= 0.50,memory dump:3f e0 00 00 00 00 00 00
i=6,s= 0.60,memory dump:3f e3 33 33 33 33 33 33
i=7,s= 0.70,memory dump:3f e6 66 66 66 66 66 66
i=8,s= 0.80,memory dump:3f e9 99 99 99 99 99 99
i=9,s= 0.90,memory dump:3f ec cc cc cc cc cc cc
i=10,s=1.00,memory dump:3f ef ff ff ff ff ff ff
i=11,s=1.10,memory dump:3f f1 99 99 99 99 99 99
 1.10
```

第 5 章

課題 5.1 FORTRAN で書いた人はプログラム見本の `nibunho.f`, `newton.f`, `senkeihanpuku.f` の中の函数 F，G を書き直すだけなので，あっと言う間にできるでしょう．三つの方法を併せて一つにした解答見本を kadai5-1.f としてここに置きました．また，C への翻訳版も `kadai5-1.c` として置きました．唯一考慮する点が有るのは，2 分法でいうと区間 $[A, B]$ をどう選ぶか，他の二つでいうと初期点をどう選ぶかですが，$e^x - 2x - 1$ に $x = 1$ を代入すると $e - 3 < 0$, $x = 2$ を代入すると $e^2 - 5 > 2.5^2 - 5 = 1.25 > 0$ であることが暗算でも分かるので，それぞれ $[1, 2]$, および 2 を選んでおくのが妥当でしょう．

線型反復法は傾きの選び方により，2 分法よりも収束が悪くなることがありますが，この問題でも 2 から普通に始めて，プログラム見本のようにその点での真の導関数の値に選ぶと 2 分法よりも大分遅くなります．見本プログラムの LINITR 函数の中の T の初期値はこちらになっていますが，これをコメントアウトし，1 行下の定義（ 区間 $[1, 2]$ に対する平均変化率で，導関数が分からなくても計算できるので，こちらの方が実用的）を復活させて試してごらんなさい．ずいぶん速くなるでしょ

う．kadai5-1.c の方は最初からこの傾きにしています．

反復法の初期値を 1 の方にすると 0 に収束してしまうのではないかと恐れる人が居るかもしれませんが，この場合は Newton 法だとこちらもほぼ同じ速さで求める値に収束します．2 の方だと無限大に発散するのではないかという恐れも起こり得るので，似たようなものですね．ただし線型反復法の方は 1 での導関数の真値を使ってしまうと収束しません．上述の平均変化率を使えば，ほぼ同じ速さで収束します．

何人かの人が Newton 法にひどく反復回数が多くかかっていたのですが，そんなはずはないので，7 回程度で終わらなかった人はプログラムを調べてみる必要があります．ただし収束値はみなさん正しい値でした．Risa/Asir により Newton 法で計算してみた値は，
$1.256431208626169676982737616609216326916 4\cdots$
となりましたので，Newton 法の最後の値は切り上げられているようですね．

実はこの解答例では，2 分法にプログラム見本と少し違う書き方をしています．それは，プログラム見本の方は

```
IF (FX*FA.LE.0.0D0) THEN
    B=X
ELSE
    A=X
END IF
```

という書き方でしたが，ここでは，

```
IF (FX*FA.LT.0.0D0) THEN
    B=X
ELSE IF (FX*FB.LT.0.0D0) THEN
    A=X
ELSE
    GO TO 300
END IF
```

のような書き方をしています．普通は後半の付け足しは無駄な計算時間を生じさせるのですが，もとのプログラム見本のままだと，実際には収束しているのに止まらなくてループを全部回ってから終了します．kadai5-1debug1.c を実行してみてください．このプログラムでは止まらない事実をはっきりさせるためにループの限界を少し大きい 60 にしていますが，これは 100 にしても同じことです．この出力データを観察すると，

```
x=07 92 6b 9c 57 1a f4 3f, y=08 92 6b 9c 57 1a f4 3f
```

になった後は $z=\dfrac{x+y}{2}$ を作っても，後者と一致してしまい，$[x,y]$ を $[x,z]$ に取り替えても区間は半分にならず，元と同じに止まっていることが見て取れます．理論的には i=54 あたりでアンダーフローして区間の長さが 0 になって終了してもよさそうなのに，止まらない理由はこういうことだったのです．これは著者もやってみるまで気づきませんでした．それで解答例としては 2 分法の条件判断を最も正当なものと取り替えておきました．

さてこの超越方程式の解は無理数です．実際，もしある有理数 $x=\dfrac{q}{p}$ に対して

$$e^{q/p}=2\frac{q}{p}+1$$

となったら，両辺を p 乗すると $e^q = (2\frac{q}{p}+1)^p$ の右辺は有理数ですから，e は代数的数となってしまい，“e は超越数である” という周知の事実に反します．ということは，理論的には区間縮小法の途中で区間の端の値が零点と一致することは無いはずなので，条件判断を最初に間違えて書いていたように

```
IF (FX*FA.LT.0.0D0) THEN
    B=X
ELSE
    A=X
END IF
```

としておいても，少なくとも最初に書いたものと同じ結果になるはずと思ってしまうのですが，やってみるとこちらの方は 1.256431208626172 という，少し大きい値を示して停止します．これも `kadai5-1debug2.c` を用いて出力値を検討してみると，i=49 で f(z)=0 となり，そのため零点をはずしてしまって右の区間に移り，そこからは意味の無い等分を繰り替えして i=49 の時点での右端だった値に収束したという訳です．この場合も i=53 で $\frac{x+y}{2}$ が y と一致してしまうのですが，右の区間を選択しているので，次の段階では区間長が 0 となって，皮肉なことにここでちゃんと停止します．一般に反復法などでは，最後の桁まで合わせようとすると，振動して止まらなくなることもあるので，少し大き目の epsilon で収束判定をしているのですが，2 分法なら大丈夫だろうと収束判定を厳しくしてみたお陰で，nibunho.f のバグに気づきました (^^;. このプログラム例では，収束判定を `x = y`, すなわち，区間の長さが 0 にアンダーフローするという条件で行っていますが，これを `y-x < 1.0e-16` としても同じです．それぞれ上記デバッグプログラムの収束判定をコメントアウトしてある方と取り替えて実行し確認してみてください．

課題 5.2　1 階微分の中心差分は 2 次の近似式なので，h に関する漸近展開は h^2 の級数となります．なので Richardson 加速は，二つ目に説明した方法を用いるのがよいのですが，演習ではほとんどの人が 1 階差分を加速するプログラム見本 `richardson.f` で前進差分のところを中心差分に置き換えるだけで済ませていたようです．これだと，加速法の部分が1 次の近似式用になっているので, 加速の段数が余分に一つ増えてしまい，かつ結果も少し悪くなります．実はこのプログラムは汎用に書かれていて，Richardson 加速のサブルーチンを呼ぶときの最後の引数 IA が近似式の次数対応を制御しているものなので，ここの実引数の 2 を 4 に変えるだけで良かったのです．それくらい，ソースを読んで分かるようになりましょう！ 以下は解答例の `kadai5-2.f` を実行してみた結果です．

```
 0.540302193338655
 0.540302305866464 0.540302298833476
 0.540302305868139 0.540302305868113 0.540302305428448
 0.540302305868136 0.540302305868136 0.540302305868135 0.540302305840655
 Richardson 加速法による値 :       0.540302305868136
 反復回数 :  5,  最後の H :   0.00625
 このHに対する 4 次差分公式の値 :  0.540302305840659
    真の値       :                  0.540302305868140
```

参考までに，1 次の漸近展開のプログラムのままでやると，

```
 0.540752163923102
 0.540302404328297 0.540414844226998
```

```
0.540302298837010 0.540302312023421 0.540330445074315
0.540302305867928 0.540302305428496 0.540302306252861 0.540309340958225
0.540302305868180 0.540302305868172 0.540302305840693 0.540302305892214
   0.540304064658716（前行の続き）
Richardson 加速法による値 :        0.540302305868180
反復回数 :  6,  最後の H :   0.003125
このHに対する 4 次差分公式の値 :   0.540302305866428
   真の値       :                    0.540302305868140
```

となり，反復回数が 1 回多い上に少し悪い値になっています．

4 次の近似式との比較には，二つの観点があります．一つは，加速法の方で最後に用いられた h と同じ値で 4 次の近似式を適用してみるというものです．この方法はもとの近似式が一発で計算できる差分公式でなく，台形公式のように重い計算の場合に, 同じ計算量でどれだけ真の値に迫れるかという意味があります．ただし厳密には，全計算量を比較しなければならないので, Richardson 加速の方には, 少し余分な計算が有る分, 対応する 4 次の近似式の h を小さくしてやらないと釣り合いません．本問のように，一発で計算できる差分公式の場合は，計算時間は Richardson 加速の方が圧倒的に不利ですが，h を小さくすると計算量がどんどん増えて行く数値積分公式の場合は，台形公式の加速と，同じ桁数の精度を出すための 4 次の公式 (Simpson 公式) とで計算量を比較することは意味があります．

二つ目の立場は, 計算量ではなく精度の優劣を比較するもので, 両者が与える最良の近似値のどちらがより真の値に近いかというものです。第一の立場で見ると Richardson 加速の方が圧倒的に有利であることが, 上の結果から分かります．二つ目の真値に迫る度合については，プログラムの出力を少し修正して問題の箇所を詳しく見ると，下の表のようになっています．最後の 2 桁が桁落ちのために振動しており，Richardson 加速の結果より真値に近いときもありますが，真値が分からない人にそれを拾い出す手段はありません．従って，いずれの観点からも Richardson 加速の方がすぐれていると結論されます．

H	4 次差分公式の値
0.002000000000000	0.540302305867842
0.001900000000000	0.540302305867887
0.001800000000000	0.540302305867937
0.001700000000000	0.540302305867992
0.001600000000000	0.540302305868055
0.001500000000000	0.540302305868077
0.001400000000000	0.540302305868101
0.001300000000000	0.540302305868073
0.001200000000000	0.540302305868101
0.001100000000000	0.540302305868135
0.001000000000000	0.540302305868027
0.000900000000000	0.540302305868060
0.000800000000000	0.540302305868101
0.000700000000000	0.540302305868049
0.000600000000000	0.540302305868102
0.000500000000000	0.540302305868176
0.000400000000000	0.540302305867917
0.000300000000000	0.540302305868226
0.000200000000000	0.540302305868103
0.000100000000000	0.540302305868474

なお，“4 次の差分公式はたくさん計算しなければならないので不利” と書いていた人が何人か居ましたが，いつも真値から離れたところまで計算して見せていたのは教育的配慮で，実際に使うとき

は，最良の h と想定されるところのちょっと先まで計算して止める (あるいは最初からそのような h だけで計算した値を用いる) ので，いつでも全部計算する必要はありません．なのでこのコメントは的外れです．

今回計算してみて，計算機室の計算結果が昔のものと微妙に異なっているのに気づきました．コンパイラや OS で計算結果に差があるようです．Richardson 加速の計算値は，みなさんのに合わせて現在のものの計算結果にしていますが，最後の 4 次の公式の揺れ具合は昔計算したもののままなので，自分で実験したときにはこれとは異なった値が出るかもしれません．(傾向は同じでしょうが.) これは恐らく丸め誤差の出方の差だろうと思われます．

課題 5.3　$f'(a) \neq 0$ のときの Newton 法の収束の速さを見た計算と同様，漸化式を評価すると

$$x_{n+1} - a = x_n - a - \frac{f(x_n)}{f'(x_n)} = -\frac{f(x_n) - (x_n - a)f'(x_n)}{f'(x_n)}.$$

ここで，$f(a) = f'(a) = 0$ を用いてわざとこれらを引くと，

$$\begin{aligned} \text{分子} &= f(x_n) - f(a) - f'(a)(x_n - a) - (x_n - a)(f'(x_n) - f'(a)) \\ &= (x_n - a)^2 \frac{f''(\xi)}{2!} - (x_n - a)^2 f''(\eta) \end{aligned}$$

ここで Taylor 展開の Lagrange 剰余を用いました．ξ, η はいずれも a と x_n の間のある値です．仮定により $f''(a) \neq 0$ なので，任意に固定した $\varepsilon > 0$ に対して $\delta > 0$ を十分小さく選んで，$x = x_n, \xi, \eta$ 等が a の δ-近傍に入る限り

$$(1 - \varepsilon)|f''(a)| < |f''(x)| < (1 + \varepsilon)|f''(a)| \tag{5.3a}$$

が成り立つようにできます．(ここで簡単のため $f''(x)$ の連続性を使いました.) すると，

$$\begin{aligned} |\text{分子}| &\leq (x_n - a)^2 |f''(\eta)| - \frac{1}{2}(x_n - a)^2 |f''(\xi)| \\ &\leq (1 + \varepsilon)|f''(a)|(x_n - a)^2 - \frac{1}{2}(1 - \varepsilon)|f''(a)|(x_n - a)^2 = \frac{1}{2}(1 + 3\varepsilon)|f''(a)|(x_n - a)^2 \end{aligned}$$

同様に，

$$\text{分母} = f'(x_n) - f'(a) = f''(\zeta)(x_n - a)$$

ここで ζ も a と x_n の間のある値です．よって

$$|\text{分母}| \geq |f''(\zeta)||x_n - a| \geq (1 - \varepsilon)|f''(a)||x_n - a|$$

これらの評価を分母分子に代入すると，

$$|x_{n+1} - a| \leq \frac{\frac{1}{2}(1 + 3\varepsilon)|f''(a)|(x_n - a)^2}{(1 - \varepsilon)|f''(a)||x_n - a|} \leq \frac{1}{2}\frac{1 + 3\varepsilon}{1 - \varepsilon}|x_n - a|$$

$\varepsilon > 0$ を十分小さくとれば，この係数

$$\lambda := \frac{1}{2}\frac{1 + 3\varepsilon}{1 - \varepsilon} < 1 \tag{5.3b}$$

となるので，これから，初期値を十分真の解に近く (具体的には, (5.3a), (5.3b) が成り立つように) 選んで Newton 法の手続きを行えば，近似列は真の解に収束すること，ただし，収束の速さは基本的な1 次の収束しか保証されないことが分かります．従ってこの場合は，わざわざ Newton 法を用

いても, 2 分法や線形反復法などに対して優位さは無いことになります. 上の評価はぎりぎりなので, たまたま 1 次収束しか出せなかったということは無いはずですが, 念のために, 例えば $f(x) = x^2$ の零点 0 を $x_0 > 0$ から出発して Newton 法の手続きで近似計算してみると,

$$x_{n+1} = x_n - \frac{x_n^2}{2x_n} = \frac{1}{2}x_n$$

で, 確かにちょうど 1 次の収束となっていることが分かります.

第 6 章

課題 6.2 おおむね Newton 法でよいのですが, $x = 0$ の近くで解くべき方程式のグラフが x 軸に接してしまうため, Newton 法の収束が悪くなります. これをどう防ぐかが面白いところです. 解答例の `kadai6-2.f` では $y = \dfrac{\sin x}{x} \fallingdotseq 1 - \dfrac{x^2}{6}$ から得られる近似解

$$x = \sqrt{6}\sqrt{1-y}$$

従って, もう少し近似を進めた

$$y = \frac{\sin x}{x} \fallingdotseq 1 - \frac{x^2}{6} + \frac{x^4}{120}$$

から

$$x = \sqrt{6}\sqrt{1 - y + \frac{x^4}{120}} = \sqrt{6}\sqrt{1 - y + \frac{36(1-y)^2}{120} + O(1-y)^3}$$
$$= \sqrt{6}\sqrt{1 - y + \frac{3(1-y)^2}{10} + O(1-y)^3} = \sqrt{6}\sqrt{1-y}\Big\{1 + \frac{3}{20}(1-y) + O(1-y)^2\Big\}$$

を用いています. この式は $|1-y| < 0.000001$, すなわち $|x| < 0.0025$ では打ち切り誤差が $O((1-y)^{5/2}) \fallingdotseq 0.000000000000001$ 以下になるので, 倍精度でほぼ満足すべき答を返すと思われます. なお, もちろんこの問題も真の解は無いので, 元の函数に代入して X に戻るかどうかで正しさをチェックしています.

第 7 章

課題 7.1 前半は 7.4 節に書いた注意事項を確認するのが内容なので, プログラム見本 `array2d.f`, `array2d.c`, `array2sub.f`, `array2sub.c` を使ってご自身でやってください. 演習ではその場でやってもらい, 考察を提出してもらいました.

後半は, 2 重ループを用いて 2 通りの方法で配列のコピーを実行し, その実行時間を計る問題です. 時間の計測は使う機械に依存するので, やり直してもあまり意味は無いと思い, 昔の実習室でやってみたときのデータをそのまま載せておきます. 時間の値自身ではなく, 相互比較のみ意味があります. 用いるプログラムは, `kadai7-1a.f`, `kadai7-1b.f` 及び, `kadai7-1a.c`, `kadai7-1b.c` で, サイズ 700 x 700 の 2 次元配列のコピーを, 添え字を i, j の順とするとき, いずれも a の方が j を先に (i.e. 内側のループで), b の方が i を先に回して 500 回繰り返すというものです. 学科の計算機演習室の機械がインテルマックだったときの実行結果は

```
kadai6-1a.c:        kadai6-1b.c:        kadai6-1a.f:        kadai6-1b.f:
real   0m1.337s     real   0m3.524s     real   0m4.483s     real   0m1.190s
user   0m1.322s     user   0m3.508s     user   0m4.451s     user   0m1.173s
sys    0m0.012s     sys    0m0.012s     sys    0m0.021s     sys    0m0.012s
```

となり, 出力形式は Linux の time 函数のそれに似ています．なお，その 2 年前は，演習室の機械は PowerPC のマッキントッシュでした．参考までにそのときの実行結果も示しておきます．

```
rep6-1a.c:  3.390u 0.030s 0:03.42 100.0%   0+0k 0+0io 0pf+0w
rep6-1b.c: 26.940u 0.040s 0:26.98 100.0%   0+0k 0+2io 0pf+0w
rep6-1a.f: 23.760u 0.020s 0:23.75 100.1%   0+0k 0+2io 0pf+0w
rep6-1b.f:  7.890u 0.040s 0:07.92 100.1%   0+0k 0+1io 0pf+0w
```

で FreeBSD 方式の出力でした．これらの数値は，

☆ user あるいは u がユーザータイム (ユーザの書いた函数を実行していた CPU 時間 (秒)

☆ sys あるいは s がシステムタイム (システムライブラリ函数を実行していた CPU 時間

☆ real あるいは3 番目の記号無し数字が全経過時間

です，昔の FreeBSD 方式では，更に % 付きの数字で

☆ このプログラムの CPU 占有率

も表示されており，リプレース前もデュアル CPU だったことも分かります．出力書式は違っても基本的な速さのデータは比較可能で，リプレースでかなり実行速度が速くなっていることが分かります．

これらの結果を見ると，いずれにおいても，C では j を内側で回したものが，FORTRAN では i を内側で回したものがずっと高速になっており，予想通りの結果と言えます．これらの数値は，CPU の占有率により，実行する度に変わりますが，傾向としては同じでしょう．

なお，こういうプログラムは，画面に何かを出力したりすると余分な計算時間がかかるので，できるだけコピー以外のものを省略する方が良いのです．C 言語の gettimeofday 函数を直接呼び出してコピーの前後に挟み，コピーだけの実行時間を計測すればメッセージなどを入れても大丈夫ですが，ここでは省略します．(後でそういう例もやるかもしれません．)

この課題は，小さな配列でやると違いがよく分からず，誤った結論を導きかねません．実際，実習では最所のうち，プログラム見本の `gauss.f` (行列のコピーというサブルーチンがそっくり含まれていることを発見した人は多かったと思います) や `gauss.c` あたりをそのまま使い 3 x 3 ぐらいの行列でやってみて，“ほとんど差は無いよ” と言っていた人が居ましたが，演習中にも注意したように，配列が小さいと，全体がキャッシュメモリに収まってしまい，アドレスの再計算の負担も小さいので，隣の番地も，一行分離れたところもほぼ同じ時間でアクセスできてしまい，この課題の趣旨には合いません．

なお，リプレース前の計算機室の機械では，この実験をしてみたら， g77 の方は大丈夫でしたが gcc にバグがあり，配列のサイズを 800 x 800 にすると，コンパイルはもちろんできるのですが，実行した途端にセグメンテーションフォールトを起こしました．今回のリプレース直後の僕の実験では gcc でも 1000 x 1000 まで大丈夫なようでしたが，リプレース前との比較のため，プログラム見本はそのままにしてあります．すなわち，配列のサイズを 700 x 700 にし，その代わりに反復実行するという手を使っています．これで FORTRAN と C の比較も可能になっています．

今回もレポートの中に，C の方で “1000 x 1000 にしたらセグメンテーションフォールトを起こした，という報告が有りましたが，gcc のバグなのか，その人のプログラムのバグなのか，どちらでしょ

う? 配列を少しずつ大きくしてゆき，どこで落ちるか調べてみてください．キャッシュが大きくなっているので，本当は大きな配列で1 回やるのと，小さな配列で何回もやるのとでは，同じではないのですが，動かない場合は仕方ないですね．

配列のサイズに言及せずに実行時間だけ書いている人が居ましたが，それでは他人の参考になる報告になりません．気をつけましょう．

演習中に，“コピーって emacs の画面でコピペすること？” という可愛らしすぎる質問が出たのは論外として，“インターネットで検索したら memcopy とか memmove とかがヒットしたけど，これを使うの？” という質問は少し内容があるので答えておきますと，これらは，二つの配列のアドレス間で記憶内容を機械語により一気に DMA 転送するので，並び方は無関係です．従って，これを使うのは今回の問題の趣旨には合いません．しかし，使える環境では問題なくこの方が高速になります．参考までに，アセンブラで書かれた libc のライブラリ函数 memmove を用いて2 次元配列をまるごと一気にコピーするプログラム `rep7-1c.c` の実行時間を載せておきます．現在の計算機室の機械では，

```
real    0m0.451s
user    0m0.440s
sys     0m0.008s
```

旧機での実行結果も

```
rep7-1c.c:  1.350u 0.020s 0:01.37 100.0%   0+0k 0+0io 0pf+0w
```

と，いずれも圧倒的な速さです．これも機械の更新で速くなっていますね．しかし，これが使えるのは二つの 2 次元配列の型が一致しているときだけであることに十分注意しましょう．

課題 7.2 `gauss.c` はプログラム見本の方に置いてあります．

課題 7.3 `det-inv.c` は `det-inv.f` とともにプログラム見本の方に置いてあります．

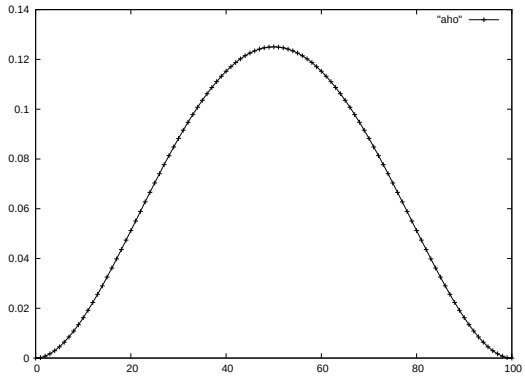
課題 7.4 `bdryvp.f`, `bdryvpxg.f` から函数を書き直すだけなので，やさしかったでしょう．みなさん gnuplot は問題なく使えたようですね．`bdryvpxg.f` の方は，コンパイルの仕方が分からない人が多かったのですが，それを見越してちゃんとソースファイルの頭に，やり方を書いておいたのです．これからのことも有りますので，メイクするまえに，ソースファイルにざっと目を通す週間を付けましょう．でないと，ソースだから安全だと思って，実はメイクしたらウィルスを実行するものができて自動的に活動されてしまうようなものにひっかかったりするかもしれませんよ．`(*^^*)`（ ただし，`Xlib` の描画ライブラリ `libX11.so` の置き場所が最近の Linux では `/usr/lib` の直下になっており，そういう環境では`-L/usr/X11R6/lib` というコンパイルオプションは不要です.）

モデル解答のプログラムは，`kadai7-4a.f` が $f(x) = 1$, `kadai7-4b.f` が $f(x) = -2 + 12x - 12x^2$ に対応し，解析解はそれぞれ $x(1-x)$, $x^2(1-x^2)$ です．真の解と数値解の差が分かるように，分割数はそれぞれ, `N = 6`, `N = 9` という荒いメッシュにして解の値を打ち出しているのですが，一つ目の方が真の解と驚異的に一致していることが分かります．後の議論（ 2 階中心差分の公式誤差評価と第 12 章で論じられる逆行列のノルム評価の定理 12.4）から示せるように，この近似解の精度は一般に $O(1/N^2)$ で，二つ目は確かにそうなっていることが実験から分かります:

N	H	中央における数値解の値	中央おにおける真の解の値
10	0.10000	0.065000000000000	0.062500000000000
100	0.01000	0.062525000000000	0.062500000000000
1000	0.00100	0.062500250000011	0.062500000000000
10000	0.00010	0.062500002500825	0.062500000000000

(この規則性を見ると，丸め誤差を無視すれば計算しなくても理論的に出せそうですね `(^^;)` では一つ目の合いすぎる説明ができますか？ヒントは，2 階の中心差分の公式誤差の評価にあります．

なお，プログラム見本の `bdryvpxg.f` は，グラフの形をこんもりさせるため，右辺の函数を 2 倍しているので，真の解も $2x^2(1-x^2)$ であることにご注意ください．`bdryvp.f` もこれに合わせてあります．`bdryvp.f` の画面出力データを教科書に示した方法でファイルに落とし，それを gnuplot に食べさせてグラフにしたものと，`bdryvpxg.f` の描写指令を Postscript ファイルとして出力したものを並べて載せておきますので，チェック用に使ってください．



gnuplot は多機能ですが，ここでは何もしていないのでスケールはめちゃめちゃだし，しかも x 軸の目盛はデータ数になっています．いろいろとオプションをつければきれいに描けるようになるので研究してみてください．XLib の方も座標軸などを追加できますが，次の章でやるのでこの段階ではこれくらいで十分でしょう．描画の方は画面では真の解を白で表示していましたが，ここは黒にしています．全般的に数値解の方が若干小さくなっていますが，出力された数値を見ると中央の値は逆に数値解の方がわずかに大きくなっていることが分かります．

第 8 章

課題 8.1　この問題は実行して感想を書くだけだったので，簡単だったでしょうが，`euler.f` をそのまま使うのはやや不便です．レポートの解答を作るのに便利な書き換え `kadai8-1.f` を作ってみました．これくらいは自分でもさっとやれますね．

誤差の影響は，教科書にも書かれているように，理論的には総誤差 $h + \dfrac{\varepsilon}{h}$ のうちの第 1 項の打ち切り誤差と第 2 項の丸め誤差が釣り合う辺り，すなわち $h = \sqrt{\varepsilon} = 10^{-8}$ あたりで最良となるはずですが，計算結果

h の値	$y(1)$ の値
0.100000000000000	2.593742460100000
0.010000000000000	2.704813829421526
0.001000000000000	2.716923932235896
0.000100000000000	2.718145926825227
0.000010000000000	2.718268237174495
0.000001000000000	2.718280469319472
0.000000100000000	2.718281692544068
0.000000010000000	2.718281814868483
0.000000001000000	2.718281827099923
0.000000000100000	2.718281828321504
0.000000000010000	2.718281828284996

で，確かに正しい桁数が h とともに延びて行くこと，すなわち 1 次の近似であることが読み取れます．しかし，理論よりはもう一つ先で最良となっていますね．このあたりの原因をさぐるのは難しいことですが，誤差のオーダーだけでなく評価の定数因子にも影響するので，これくらいなら理論

通りだと思ってもよいでしょう．この計算は昔の計算機室の機械でここまでやるのはかなり大変でした．グラフィックや途中経過の出力など極力抑制し，コンパイル時に -O2 のオプティマイズをかけると，何とか実行可能になったはずです．

なお，この真値 e のことを “自然対数” と読んでいる人が結構居ましたが，正しくは，“自然対数の底” です．対数は函数の名前ですね．

課題 8.2　x^2+y^2 という量を解軌道に沿って時間微分する，より初等的に言うと，$x=x(t), y=y(t)$ が連立微分方程式の解のときに $x(t)^2+y(t)^2$ を独立変数 t で微分してみると

$$\begin{aligned}\frac{d}{dt}(x^2+y^2) = 2x\frac{dx}{dt} + 2y\frac{dy}{dt} \quad &= 2x\times y + 2y\times(-x) \quad (\text{方程式を代入した})\\ &= 2xy - 2xy = 0\end{aligned}$$

解析解の場合はたったこれだけの計算です！

次に Euler 法の近似スキームの場合には，$h=\dfrac{2\pi}{N}$, として $(x,y)=(x_0,y_0)=(1,0)$ から出発することを考えると，n ステップ進んだときの位置を (x_n,y_n) と書けば，

$$x_{n+1} = x_n + hy_n, \quad y_{n+1} = y_n - hx_n,$$

従って

$$x_{n+1}^2 + y_{n+1}^2 = (x_n+hy_n)^2 + (y_n-hx_n)^2 = (1+h^2)(x_n^2+y_n^2)$$

つまり，1 ステップごとに原点からの距離が $(1+h^2)^{1/2}$ 倍になるので，一周して戻ると

$$(1+h^2)^{N/2} = (1+h^2)^{\pi/h} = \{(1+h^2)^{1/h^2}\}^{\pi h} \doteqdot e^{\pi h} \doteqdot 1+\pi h$$

つまり，h の 1 次のオーダーで 1 周毎に半径が延びていく．これは無視できない大きさなので，数値解のグラフは見るからに渦巻き状に広がってゆくのです．

課題 8.3　前問と同様の記号を使うと，

$$x_{n+1} = x_n + hy_n, \qquad y_{n+1} = y_n - hx_{n+1}, \quad \text{従って} \quad y_{n+1} = (1-h^2)y_n - hx_n$$

すると，ヒントに示した量を計算してみると

$$\begin{aligned}&x_{n+1}^2 + hx_{n+1}y_{n+1} + y_{n+1}^2\\ &= (x_n+hy_n)^2 + h(x_n+hy_n)\{(1-h^2)y_n - hx_n\} + \{(1-h^2)y_n - hx_n\}^2\\ &= (1-h^2+h^2)x_n^2 + \{2h + h(1-h^2-h^2) - 2h(1-h^2)\}x_ny_n + \{h^2 + h^2(1-h^2) + (1-h^2)^2\}y_n^2\\ &= x_n^2 + hx_ny_n + y_n^2\end{aligned}$$

となって変化していない．つまり，数値解の軌跡は常に一定の楕円 $x^2+hxy+y^2=1$ の周上にあるので，軌道は発散も縮退もし得ない．さらに，この楕円は h を小さくしていくと単位円 $x^2+y^2=1$ に近づく．

なお，微分方程式の不変量を直接調べると

$$x_{n+1}^2+y_{n+1}^2 = (x_n+hy_n)^2+\{(1-h^2)y_n-hx_n\}^2 = (1+h^2)x_n^2+\{2h-2h(1-h^2)\}x_ny_n+\{h^2+(1-h^2)^2\}y_n^2 = (1+h^2)x_n^2$$

となり，増えているのか減っているのかはっきりしない．上の保存量の式は大きめの h で数値実験したときに見えてくる楕円から推測できる．

課題 8.4 (略)

課題 8.5 課題 8.1 と同様にプログラム見本 `rungekutta2.f` を `kadai8-5a.f` に，`rungekutta4.f` を `kadai8-5b.f` に修正して実行してみた結果を載せます．まず，2 次の Runge-Kutta の方は

h の値	$y(1)$ の値
0.100000000000000	2.714080846608224
0.010000000000000	2.718236862559957
0.001000000000000	2.718281375751763
0.000100000000000	2.718281823928888
0.000010000000000	2.718281828413745
0.000001000000000	2.718281828458639
0.000000100000000	2.718281828459635
0.000000010000000	2.718281828457723
0.000000001000000	2.718281828459139

こちらについても，総誤差 $h^2 + \dfrac{\varepsilon}{h}$ における二つの項の釣り合いから予想される $h = \varepsilon^{1/3} = 10^{-5}$ より二つほど先で最良となっています．

次に 4 次の Runge-Kutta の方は，

h の値	$y(1)$ の値
0.100000000000000	2.718279744135166
0.010000000000000	2.718281828234403
0.001000000000000	2.718281828459033
0.000100000000000	2.718281828459038
0.000010000000000	2.718281828459108
0.000001000000000	2.718281828459542
0.000000100000000	2.718281828458236
0.000000010000000	2.718281828457899
0.000000001000000	2.718281828458934

で，最良の h の理論値は，総誤差 $h^4 + \dfrac{\varepsilon}{h}$ における二つの項の釣り合いから予想される $h = \varepsilon^{1/5} = 10^{-3}$ とほぼ合っています．これらの計算から，$h = 10^{-7}$ のときに，丸め誤差の影響が解の 10^{-12} あたりの桁に及んでいることが分かりますが．それ以後は必ずしも 1 桁ずつ増えている訳ではなく，むしろ盛りかえしたりしています．丸め誤差は打ち消し合うこともあるので，これを精密に捉えるのは非常に難しいのです．

課題 8.6 (略)

課題 8.7 (略，プログラム見本 `rungekutta4.c` 参照．これは `xgrc.o` をリンクして画面に解曲線を表示するようになっています．)

第 9 章

課題 9.1 (略)

課題 9.2 (略，実行画面は教科書にはモノクロームで載せているので，自分で実行して鑑賞することができない読者のため，本書のサポートページのトップにカラーの原版を載せておきました．)

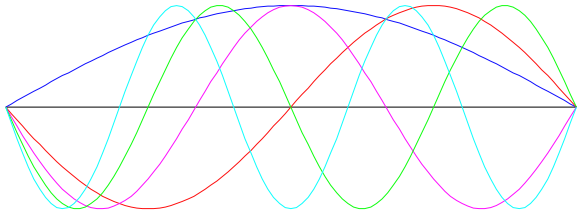
課題 9.3 (略，同上)

課題 9.4 (略，同上)

第 10 章

課題 10.1　(略，解答例は `kadai10-1.c` 参照．)

課題 10.2　(略) `sl-eigenv.f` の実行結果のカラー出力画面を Postscript にしたものだけここに載せておきます．固有関数は単にノルムを 1 に正規化しているので，乱数で出力された元データに依存して正負の符号だけのあいまいさがあり，教科書の挿絵とは大分上下がひっくり返っています．



課題 10.3　(略，出力図はこのサポートページのトップに置きました．)

第 11 章

課題 11.4　時間に依存する非斉次項を追加する問題です．プログラム見本の `heat.f` を修正して用います．解答例は `kadai11-1a.f` として置きました．自分で頑張る人の参考に，`heat.f` からの修正のポイントを示しておきます．

1. 考察する区間を $[0,1]$ から $[-1,1]$ に変更する．これは `XMIN` の値を変えるだけの簡単なものです．
2. 初期函数の変更．
3. 右辺の函数の実装．これは，$(1+x^2)e^{-t}$ をプログラムするだけなので，どうということはないでしょう．
4. 差分法の修正．方程式が $\dfrac{\partial u}{\partial t} = \nu\dfrac{\partial^2 u}{\partial x^2} + f(x,t)$ に変わるので，差分化は，

$$\frac{u(x,t+k)-u(x,t)}{k} = \nu\frac{u(x+h,t)+u(x-h,t)-2u(x,t)}{h^2} + f(x,t)$$

と新たな項が加わり，これに対応して漸化式の右辺にも新たな項が加わり，

$$u(x,t+k) = \Big(1-\frac{2\nu k}{h^2}\Big)u(x,t) + \frac{\nu k}{h^2}u(x+h,t) + \frac{\nu k}{h^2}u(x-h,t) + kf(x,t)$$

となります．従ってこれを離散化すると，f の値を $(a+hi,kj)$ で一々計算して加えなければなりません．`heat.f` では，kj の値は保存されておらず，また $a+hi$ を毎度計算するのはスピードの点で問題なので，元のプログラムでは必要なかった t,x という変数を追加し，これらをそれぞれ 0 や `XMIN` から増加させてゆく必要が生じます．

さて，この問題では，真の解が分かっているので，それと数値解を比較することができます．このためのプログラム見本を `kadai11-1b.f` として用意しました．もとのプログラム見本は，空間メッシュが 80 等分なので，このままでは真の解と数値解のグラフを重ね描きしても違いが見えません．

そこで，こちらは分割数 N を入力するようにしてあります．$N = 4$ 等分くらいに思い切り減らして重ね描きして実験してみてください．もちろん，比較はグラフにせず，数値で出力してもよいのです．プログラム `kadai11-1b.f` 中に真の解との差を出力する行が埋め込んでありますので，グラフを二つ重ねて描くことができなかった人は，この行のコメントを外して実行してみてください．N が大きいときは，差のベクトルをそのまま表示せず，成分の絶対値の最大値だけを出力すると分かりやすいでしょう．これを誤差の最大値ノルムと呼びます．

考察としては，解析解 $(1 - x^2)e^{-t}$ を見ると，上に凸な同じ放物線状のグラフが，時間の経過とともに全体として指数的に低くなってゆくことが予想されますので，数値解の挙動もそうなっていることを確認できれば成功です．ただしこのような書き方には，少し注意が必要です．この講義で今まで強調してきたように，計算機実験は必ずしも理論の予想通りになるとは限りません．実験を客観的に見るようにし，理論から予想される状況に無理やり合わせてしまうことは避けましょう．この問題では，講義の見本プログラムと同様，$\lambda = 0.5$ で安定な数値解が得られますが，この点も実験で確かめましょう．